An Implementation of a Combining Network for the NYU Ultracomputer

Susan Dickey, Richard Kenner, Marc Snir[1]

Ultracomputer Research Laboratory
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012

Ultracomputer Note #93

January, 1986

ABSTRACT

The NYU Ultracomputer architecture, a shared memory MIMD design composed of thousands of processing elements, requires a high-performance interconnection network between its processors and memory modules. This network must be capable of merging concurrent requests to the same memory location from multiple processors. We present an implementation of such a network for a 32-processor configuration with emphasis on the mechanisms used to merge these concurrent requests.

1. Introduction

The NYU Ultracomputer architecture (described in detail in Gottlieb *et al.* [83], Edler *et al.* [85], and the references therein) is a shared memory MIMD design capable of supporting thousands of processing elements. Each processing element is augmented by a fetch-and-add (F&A) operation. The function F&A(V, e) indivisibly increments the (shared) variable V by the value of the expression e and returns the old value of the memory location specified by v. Concurrent F&A operations are required to have the same effect as if executed in some (unspecified) serial order. This F&A operation is a powerful interprocessor communication mechanism which can be used to eliminate most critical sections and thus to permit highly concurrent execution of algorithms, such as queue insertion and deletion, that are important in operating systems and parallel applications (Gottlieb and Kruskal [81]).

To approximate the performance of an idealized parallel processor (dubbed a "paracomputer" by Schwartz [80] and a WRAM by Borodin and Hopcroft[81]) where all processors can simulataneously access memory in a single cycle, a high-

[1]This work was supported in part by the Army Research Office under grant number 20391-MA-F, and in part by the National Science Foundation under grant number DCR-8413359.

performance network is required to connect the processing elements (PE's) with the memory modules (MM's) comprising the shared memory. This message switching network, which has the topology of Lawrie's [75] Ω-network, is described in Dickey *et al.* [85] and has the following properties:

- The network contains N log N identical nodes where N is the number of PE's.
- Network bandwidth is linear in N.
- Latency, i.e., memory access time, is logarithmic in N.
- Messages are divided into packets which are transmitted in successive cycles. If no routing conflicts exist, at every cycle each switching node can receive an input packet at each input port and transmit a packet at each output port.
- Switch settings are maintained only for a single message (the network is message switched).
- Routing decisions are local to each switch; thus routing is not a serial bottleneck and is efficient for short messages.
- Switching nodes contain queues used to store messages which cannot be immediately transmitted due to routing conflicts.
- Concurrent access by multiple PE's to the same memory cell suffers no performance penalty.

It is the last feature of our network on which this paper will focus. The motivations behind this feature are clear. If requests from multiple processors to the same location were to be processed serially at a memory module, we would be adding to the MM's the critical sections which were eliminated due to the use of the F&A operation and thus merely moving the serial bottlenecks from the software to the hardware. In fact, simulations have shown that even moderate traffic to shared coordination variables ("hot-spots") can severely degrade all memory accesses, not just accesses to the coordination variables (Pfister and Norton [85]).

Our implementation of merging multiple requests from different PE's for the same memory location ("combining") is based on the fact that, in an Ω-network, a unique path exists between each processor and each memory module. Hence each memory module can be viewed as the root of a tree having the PE's as leaves. Two simultaneous requests destined for the same memory location will meet in some switch where they can be combined into one request[2] which is passed further down the network where it might meet other requests destined for the same location and be further combined.

This paper is organized as follows: Section 2 will discuss tradeoffs that have been made in a planned implementation of our network. We will then discuss in sections 3 and 4 the protocols and specifications of the components used in this implementation of the network. Finally, section 5 will discuss the design of the

[2]In the network we will be describing, the combining is actually done at the stage following the stage at which the requests first meet.

network components with emphasis on the return path from the memory to the processors. The design of the forward path is given in Dickey *et al.* [85].

2. Network Overview

We will be describing the design of a combining switching network for a specific Ultracomputer, namely a 32-processor prototype we plan to construct over the next two years. This system will contain 32 PE's (Motorola 68020's with 68881 floating-point co-processors) and 32 MM's with each MM containing up to 4MB of memory. Since this will be our first implementation of a combining network, in some cases where the performance loss is minimal we will be trading off performance in favor of simpler logic in the network.

It has been shown (Kruskal and Snir [83]) that queueing delays increase linearly with the number of packets per message. Since we wish to keep these delays as small as possible and since the logic for a combining queue when the address spans more than one packet is quite complex (Snir and Solworth [84]), the network configuration described below will have one address packet and one or more data packets per message.

Since the systolic queue design we are using (Dickey *et al.* [85]) requires that messages contain an even number of packets[3] and the width of the data to be transmitted in each message is approximately the same as the width of the address and control information, each message will contain *exactly* two packets. The first packet will contain address and control information and the second packet will contain the data. Since this format produces an unused packet in a load request and a store acknowledgement, we can transmit a load request as an F&A of 0 without causing any overhead in the network while simplifying the switch logic[4].

The network described below does not support the combining of stores directed to the same memory location: combining of stores directed to different bytes within the same word requires extra logic to OR the byte select bits and, more significantly, extra logic to select the relevant bytes from both data words, according to the value of their byte select bits. Although combining stores involving the same bytes is simpler, the utility of combining stores is not deemed to justify the extra effort for this first implementation. (Heterogeneous combining of F&A's with stores is quite complex because the operations of store and F&A do not commute and thus care must be taken to avoid improperly[5] interchanging an F&A and store for the same memory location.)

---

[3]More precisely, it requires that there be an even number of cycles between successive transistions of the *queue filling* and *queue emptying* signals.

[4]Although not strictly required, the memory modules are assumed to detect the case of an F&A with data of zero and suppress the addition and store.

[5]The fundamental complexity is in the determination of which interchanges are "improper".

3. Switch Configuration and Protocols

Each switching node consists of two components: a forward path component (FPC) and a return path component (RPC). The FPC's route messages from the PE's to the MM's and the RPC's return the responses from the MM's to the PE's. Within each RPC are wait buffers (WB's) used to store combined requests while awaiting responses from the MM's.

Each of these components has six ports. There are two input ports (IP's), which connect to a previous stage and two output ports (OP's), which connect to a subsequent stage. In addition, two wait buffer output ports (WBOP's) on the FPC are connected to two wait buffer input ports (WBIP's) on the RPC of the same switch.

The construction of the queues used in the switches requires that switches distinguish odd and even cycles. Therefore, each component keeps a count modulo two of clock cycles. The cycle parity of the FPC and RPC in the same switch are identical[6] and the parity of switches at successive stages differs by one so that a cycle that is even for one switch is odd for its successor and predecessor, and vice versa. The cycle count of a component is initialized by an external signal: the first cycle where the initialization signal is low is an even cycle.

Each component starts receiving a message on an input port only at even cycles and starts transmitting a message on an output port only at odd cycles. Message transmission from a WB output port to a WB input port starts at odd cycles.

Each port consists of 35 data bits and two protocol bits: a data valid bit (DV) travelling in the same direction as the data and a data accept bit (DA) travelling in the reverse direction. In addition, input ports receive a routing (RO) bit.

The two protocol bits, in conjunction with the RO bit, regulate the transmission of messages through the network. An input port asserts the DA bit at cycle t if it is willing to accept a message during the two successive cycles ($t+1$ and $t+2$). The DV bit is asserted by an output port during the transmission of a message. The value of the RO bit at the first cycle of a message transmission indicates to which output port the message is destined. (The value of a DA bit generated at even cycles by an input port or at odd cycles by a WB input port, the value of a DV bit generated at even cycles by an output port or at odd cycles by a WB output port, and the value of a RO bit generated at even cyles by an FPC are all ignored.)

The routing bit accompanying a packet entering an FPC at the i-th stage of the network is the i-th most significant bit of the MM number; on the RPC and the WB input port it is the i-th most significant bit of the PE number. These connections are achieved by externally hardwiring the pins accordingly (no extra pin is needed at an output port while an extra pin needed at each input port).

---

[6]This implies that memory accesses must take an odd number of cycles. Alternatively, the cycle parity for the FPC and RPC of the same switch could differ, which would imply that memory accesses must take an even number of cycles.

Packets sent on the network are 35 bits wide. The first packet of a message (called the address packet) has the following format:

Bits	Purpose
0-4	MM number
5-24	address within MM
25	opcode: "F&A" or "Store"
26-30	PE number
31-34	byte select

The MM number and the address within the MM uniquely identify each word of the shared memory. The opcode is used to distiguish F&A's, which are combined, from stores, which are not. The byte select bits are used in store operations to indicate which bytes in the word are to be modified by the MM.

The PE number is used to route the memory response to the requesting processor. An alternate scheme, described in Dickey *et al.* [85], in which a single 5-bit field contains a combination of the MM and PE number could also be used. In this method, which reduces the width of network messages,[7] each component uses the high-order bit of this field to select the destination output port, then replaces this bit with the number of the originating input port and rotates the field one bit to the right. Thus, during each transit through the network, the destination address is gradually replaced by the source address. However, because the number of bits containing the destination address varies between stages, each component must know its stage number in the network to determine which bits of the address are significant in comparisons. In our initial network implementation, we will be using separate PE and MM addresses to avoid this complexity.

The second (data) packet contains a 32 bit word (bits 0-31) which, in requests to an MM, contains either the increment value for an F&A or the value to be stored for a store operation. In responses to an F&A, the data packet is used to return the old value of the memory word being read. The data packet is unused in responses to stores. The last 3 bits of a data packet are unspecified and can be used for parity or flag bits between the processor and memory.

For each combined message, an FPC must transmit to the wait buffer in an RPC sufficient information for the RPC to recognize when a response to the combined message has been received and to generate the response messages to transmit to both requesting processors. The latter condition is met by transmitting the data packet from either F&A request to the RPC.

The former condition can be met in one of two ways. The simplest is to ensure that every combinable request transmitted concurrently through the network can be uniquely identified by its address packet. The wait buffer will then perform an associative search to detect the matching request. If the design of a PE is

[7] Another advantage is that a PE need not know its own position in the network.

constrained to have at most one combinable (load or F&A) request outstanding to a single location, the combination of PE number, MM number, and address within MM is sufficient to uniquely identify each combinable request in the network.[8]

An alternative way to allow an RPC to detect that a response corresponds to a particular combined message is to have the FPC tag each combined message that leaves its output port with a stage number and a unique identifier which is an address in a (non-associative) wait buffer. This method has the advantage of eliminating some constraints on the processor[9] and the need for an associative memory in the wait buffer at the expense of additional bits in network messages. However, it requires a method of communicating available addresses within the wait buffer to the FPC and requires that each stage know its stage number. Because of these complexities, and the fact that most current microprocessors are not capable of issuing multiple outstanding requests, we have decided not to implement this method.

Thefore, the address packet of messages sent between the FPC and RPC via the WB input and output ports is also 35 bits long and has the following format:

Bits	Purpose
0-4	MM number
5-24	address within MM
25-29	PE number
30-34	second PE number

The opcode and byte select lines are not needed because only F&A's are combined by this network. The two PE number fields of the message identify the two processors whose requests combined to form this message.

4. Functional Specification

Each component receives messages from its two input ports, according to the protocol previously described, and routes them to its two output ports, selected according to the value of the RO bit associated with each message. A packet is delayed at each component for an odd number of cycles.

Two requests arriving at an FPC from the same input port can be combined if their MM number and address within MM agree and they are both F&A operations. Suppose a message of the form

---

[8]At the cost of additional bits in a message, processors could be permitted to have multiple F&A's oustanding by augmenting each message by an additional field generated by the processor to uniquely tag each such request.

[9]Note, however, that a processor can never be permitted to have both an F&A and a store to the same memory location outstanding simultaneously. Consider the case where PE_i performed a store to location n followed by an F&A to the same location. If an F&A to location n had previously been issued by, say, PE_j, this request might combine with the F&A from PE_i. If it did, the order of the store and F&A requests issued by PE_i would have been reversed.

MM#	addr	"F&A"	PE_i	xxxx	val_i

is received on input port *a* for output port *c* and a subsequent message

MM#	addr	"F&A"	PE_j	xxxx	val_j

is also received on input port *a* for output port *c*. If the first message has not already departed, the second message can combine with it. If they are combined, a single message of the form

MM#	addr	"F&A"	PE_i	xxxx	val_i+val_j

is sent to the next stage through output port *c* and a message of the form

MM#	addr	PE_i	PE_j	val_j

is simultaneously sent to the RPC though the WB output port corresponding to output port *c*. This message is stored in the RPC together with its associated RO bit.

When a response from an MM arrives at an RPC, it is compared with all the messages currently stored there. A match occurs if the opcode of the returning message indicates an F&A and the MM number, address within MM, and PE number field of the response agrees with these fields in some stored message. Since these fields uniquely identify an F&A request in the network, only one match is possible. When the RPC receives a response of the form

MM#	addr	"F&A"	PE_i	xxxx	rval

that matches a message previously stored, the response is forwarded to the corresponding output port, the previously stored message is deleted, and a new response of the form

MM#	addr	"F&A"	PE_j	xxxx	$rval+val_j$

is generated and sent through OP_k, where *k* is the routing bit of the message previously stored (the number of input port *a*).

5. Implementation

We now present a design of the forward and return path components which implements the protocols and specificatins described above.

5.1. Forward Path Component

The FPC's route memory requests (F&A's and stores) from the PE's to the MM's and are the components which detect and merge combinable requests. FPC's have two input ports which connect to earlier network stages (or a PE), two output ports which connect to subsequent network stages (or an MM), and two wait buffer output ports which connect to the RPC of the same switching node. Each output port contains a 32-bit adder used to generate the sum of the data values for two combined F&A operations.

In order for each input port to be able to accept an input packet each cycle, no two input ports may share a queue; to prevent a backlog at one output port from blocking messages to the other output port, there must be a separate queue between each input and output. Therefore, each FPC has four combine queues, each corresponding to an input/output pair.[10] We will refer to these queues as CQ_{ij}, $0 \le i,j \le 1$ where CQ_{ij} is fed by IP_i and writes to OP_j and $WBOP_j$. CQ_{ij} accepts a message at cycles $2t$ and $2t+1$ if DV is asserted at cycle $2t$ at IP_i, and the value of RO_i at cycle $2t$ is j.

These combine queues are described in Dickey *et al.* [85]. They are capable of accepting and transmitting a packet on each cycle and if a queue is empty, a packet will transit a queue in one cycle.

Each port sets its DA bit to indicate whether it has room to accept a message. Specifically, DA_i will be asserted at cycle $2t-1$ if both CQ_{i0} and CQ_{i1} have enough room to receive a message at cycles $2t$ and $2t+1$. We will construct each queue with an even number of slots and thus if there is exactly one empty slot at an odd cycle, the queue is known to be transmitting and there will thus be two empty slots at the next cycle. Therefore, a DA signal is asserted at IP_i at cycle $2t-1$ if, at that cycle, there is at least one empty slot in both CQ_{i0} and CQ_{i1}.

A queue is not permitted to transmit a message unless the subsequent stage is ready to receive it. In addition, it cannot transmit a message to the wait buffer in an RPC if the wait buffer is full. We therefore define CQ_{ij} as being *ready to send* at cycle $2t+1$ if either

(1) its top message is an "uncombined" message (to be forwarded only to an output port) and DA was received at OP_j at cycle $2t$; or

(2) its top message is a "combined" message (i.e., it has data to send to both an output port and a wait buffer output port) and DA was received at cycle $2t$ at both OP_j and $WBOP_j$.

Since both CQ_{0j} and CQ_{1j} output to OP_j, if both are simultaneously *ready to send*, they must take turns presenting their messages to the output port. Each queue records whether it or its partner was the last to send a message to the output port. We define CQ_{ij} as having the *right to send* if either $CQ_{\bar{i}j}$ is not *ready to send* or $CQ_{\bar{i}j}$ was the last to send.

If CQ_{ij} is *ready to send* and has the *right to send*, it asserts the DV signal at OP_j at cycle $2t+1$ and sends its top message through this port at cycles $2t+1$ and $2t+2$. If it is a combined message, DV is simultaneously asserted at at $WBOP_j$ and the message is simultaneously sent through that port to the wait buffer in the RPC.

---

[10]This reduces by one the number of opportunities that a message has to combine with other messages as it transits the network. There are two methods of avoiding this: At the expense of an additional stage of delay, an additional combining queue can be placed between the network and the MM's (however the combining queue adjacent to the PE will be "wasted"). Alternatively, the two combine queues connected to the same output ports can be merged into one combine queue with two inputs at the expense of a more complicated, and presumably slower, queue.

5.2. Return Path Component

The RPC routes the responses from MM's to the requesting PE's. When a response to a request previously merged by the FPC is detected, the RPC will generate an additional response so that both requesting PE's will receive responses.

Each RPC has two input ports connected to the output port of an RPC from a previous (in this case, closer to the MM's) stage (or to an MM) and two output ports connected to the input port of an RPC from a subsequent stage (or to a PE). In addition, two wait buffer input ports receive information from the FPC while allow the RPC to process responses to combined requests. One wait buffer is associated with each input port.

As shown in Figure 1, an RPC contains 2 wait buffers, WB_0 and WB_1, and 8 non-combining queues, Q_{ijk}, $0 \le i,j,k \le 1$. (As in the FPC, to enable each input to accept a packet every cycle and to prevent a blockage of one output from interfering with the other output, one queue is required for each input/output pair. In the RPC, however, inputs include both wait buffers and input ports.) Queue Q_{0ij} is fed from IP_i and writes on OP_j; queue Q_{1ij} is fed from WB_i and writes on OP_j.[11]

Messages received on IP_i with routing bit set to j on cycle $2t$ and $2t+1$ are sent to Q_{0ij} and also to WB_j where its first (address) packet is compared with the messages currently in the wait buffer. If a match is found, the wait buffer asserts its *match* line during cycle $2t+1$ and starts to send its generated response to Q_{1ij} at cycle $2t+2$. The delay from cycle $2t+1$ to cycle $2t+2$ is required so that queues Q_{0ij} and Q_{1ij} receive the first packets of their messages at cycles of the same parity.

In the same manner as in the FPC, the DA signal can only be asserted at cycle $2t+1$ at IP_i if there is at least one empty slot in the two queues Q_{0i0} and Q_{0i1}. In addition, there must be sufficient room in the two queues Q_{1i0} and Q_{1i1} for messages from WB_i corresponding to both the last message received at IP_i and the current message. Therefore, the DA bit also cannot be asserted unless WB_i does not assert *match* and there is one empty slot in each queue Q_{1ix} or WB_i asserts *match* and there are three empty slots in both[12] of these two queues.

To arbitrate between the four queues Q_{xyk} that can end data to OP_k, the RPC keeps track of when each of these queues has last sent a message. The queue Q_{ijk} has the *right to send* at cycle $2t+1$ if all queues Q_{xyk} that are not empty have sent more recently than Q_{ijk}.

If Q_{ijk} is not empty and has the *right to send* at cycle $2t+1$ and the DA signal was received at cycle $2t$ at OP_k, it asserts DV at OP_k at cycle $2t+1$ and sends its message on OP_k at cycles $2t+1$ and $2t+2$.

---

[11] A equivalent, but more complex, implementation is to replace the 4 queues attached to the same output port by one queue with four inputs.

[12] Since the destination of the message in WB_i is known on-chip at this cycle, this test can be refined to only require three empty slots in the queue that is the destination of that message.

The DA signal is asserted at cycle $2t$ at a wait buffer input port if the wait buffer will have an available slot to receive a message at cycles $2t+1$ and $2t+2$. As will be seen below, a slot in the wait buffer is capable of simultaneously receiving and transmitting a message. Therefore, a DA signal will be asserted at cycle $2t$ on $WBIP_i$ if at that cycle WB_i either does not assert the *full* signal or asserts the *match* signal. The wait buffer accepts a message at cycles $2t+1$ and $2t+2$ if DV is received at cycle $2t+1$ by WB_i.

5.2.1. Wait Buffer

The wait buffer is an associative memory that stores information sent by the FPC when combining two F&A's into a single request. The wait buffer inspects all responses from MM's and searches for a response to a request previously combined by the FPC. When it finds a response to such a request, it generates a second response to that request and deletes the request from its memory.

The structure of the wait buffer (WB) is shown in Figure 2 and the structure of each slot in the wait buffer is shown in Figure 3. The wait buffer consists of a number of message slots. Each slot consists of two registers (called the Areg and Breg), compare logic, and a controller. Each register contains 37 bits consisting of 35 data bits, a data valid (DV) bit, and, for the first packet of each message, a routing (RO) bit. The registers are connected in a circular loop of length two, and shift at each cycle. The Areg receives the address packet of a message at even cycles and the data packet at odd cycles. The opposite is true for the Breg. Packets are stored in the format they are received from the WB input port with the RO bit appended to the address packet of each message.

Each slot connects to the following buses:

- The write bus (Wbus) is used to send data to the wait buffer from the FPC; it is 37 bits wide (35 data bits, 1 RO bit and 1 DV bit) and connects to a wait buffer input port.
- The read bus (Rbus) is used by each slot for transmission of its message out of the wait buffer; it is 33 bits wide since the PE number field is not transmitted from the wait buffer.
- The key bus (Kbus) contains the search key received from the input port of the RPC; it is 32 bits wide and carries the following information:

0	5	25	30	31
MM number	address	PE number	opcode	Dv

A next-slot (NS) line is a one bit signal which is passed through all the slots in a daisy chain fashion. It is used to select which slot will receive the next message from the FPC. Each slot computes

$$NS_{out} := NS_{in} \text{ and not } empty$$

and the end of this signal, which has passed through all the slots, is the *full* line of the wait buffer.

A 32-bit adder is used to generate the second response to an F&A operation by summing the data packet received from a slot and the data packet received from an MM. It passes address packets unchanged. Since the wait buffer is required to output its response two cycles after it receives a receives a matching memory response, the adder can take a full cycle to perform its addition.

A 32-bit Dreg is connected to the input port. It will be loaded on odd cycles with the data packet of a message. It will present that packet to the adder on the next cycle, which will forward the sum to the appropriate RPC queue on the cycle after that.

Each slot is *full* or *empty* depending on the DV bit in its Areg. At each pair of cycles $2t$ and $2t+1$, each wait buffer slot simultaneously performs the following operations:

- If the slot is *empty*, DV is present on the Wbus, and NS_{in} is on,[13] the Areg is loaded with the address packet of the message at cycle $2t$. During cycle $2t+1$ the Breg receives the address packet and the Areg receives the data packet of the message.
- If the slot is *full*, the data in the Areg and the data on the Kbus are compared during cycle $2t$. If the DV bit is present on the Kbus, the operation on the Kbus is an F&A, and the MM number, address within MM, and PE number are the same in the Areg as on the Kbus, the slot asserts *match*. (Note that only one slot can detect a match because the combination of MM address, address within MM, and PE number uniquely identify each combinable request in the network.) If a match is detected on cycle $2t$, the slot will present its message on the Rbus at cycles $2t+1$ and $2t+2$. It will also be marked as *empty* during cycle $2t+1$ so that it can begin accepting a subsequent message at cycle $2t+2$.

If any slot asserts its *match* signal during cycle $2t$, the *match* signal of the wait buffer will be asserted at cycle $2t+1$. The packet presented by a slot to the Rbus on cycles $2t+1$ and $2t+2$ will be presented to Q_{1ij} on cycles $2t+2$ and $2t+3$ after being processed by the adder.

6. Packaging

We anticipate packaging the FPC and RPC on separate chips. Thus, a network connecting 32 PE's with 32 MM's will have 80 switching nodes and 160 chips. The number of I/O connections on each switch component are as follows:

[13] As a performance optimization, a slot can load its registers whenever it is *empty*. The DV bit is then set from the logical AND of the DV bit on the Wbus and the NS_{in} signal.

name	width	multiplicity	total
data lines	35	6	210
DV & DA bits	2	6	12
Routing bits	1	2	2
Clocks	2	1	2
Vdd	1	5	5
Ground	1	7	7
Init line	1	1	1
Total			239

The RPC's need two more pins (i.e. 241) as they need two routing bits on their wait buffer input ports.

7. References

A. Borodin and J. E. Hopcroft, "Merging on Parallel Models of Computation", Manuscript, 1981.

Susan Dickey, Richard Kenner, Marc Snir and Jon Solworth, "A VLSI Combining Network for the NYU Ultracomputer" in *Proc. International Conference on Computer Design.*, 1985.

Jan Edler, Allan Gottlieb, Clyde Kruskal, Jim Lipkis, Larry Rudolph, Marc Snir, Pat Teller, and James Wilson, "Issues Related to MIMD Shared Memory Computers: The NYU Ultracomputer Approach", *Proc. 12 Annual Computer Architecture Conf.*, 1985.

Allan Gottlieb, Ralph Grishman, Clyde P. Kruskal, Kevin P. McAuliffe, Larry Rudolph, and Marc Snir, "The NYU Ultracomputer—Designing an MIMD Shared Memory Parallel Computer", *IEEE Trans. Comp.* **C-32**, pp. 175-189, Feb. 1983.

Allan Gottlieb and Clyde P. Kruskal, "Coordinating Parallel Processors: A Partial Unification", *Computer*Architecture pp. 16-24, October 1981.

L.J. Guibas and F.M. Liang, "Systolic stacks, queues and counters", in *Proc. Conf. Advanced Research VLSI,* Jan. 1982.

Duncan Lawrie, "Access and Alignment of Data in an Array Processor", *IEEE*Trans. **C-24**, pp. 1145-1155, 1975.

G.F. Pfister and V.A. Norton, "'Hot Spot' Contention and Combining in Multistage Interconnection Networks", *Proc. 1985 Int. Conf. on Parallel Processing.*

J. T. Schwartz, "Ultracomputers", *ACM TOPLAS* **2**, pp. 484-521, 1980.

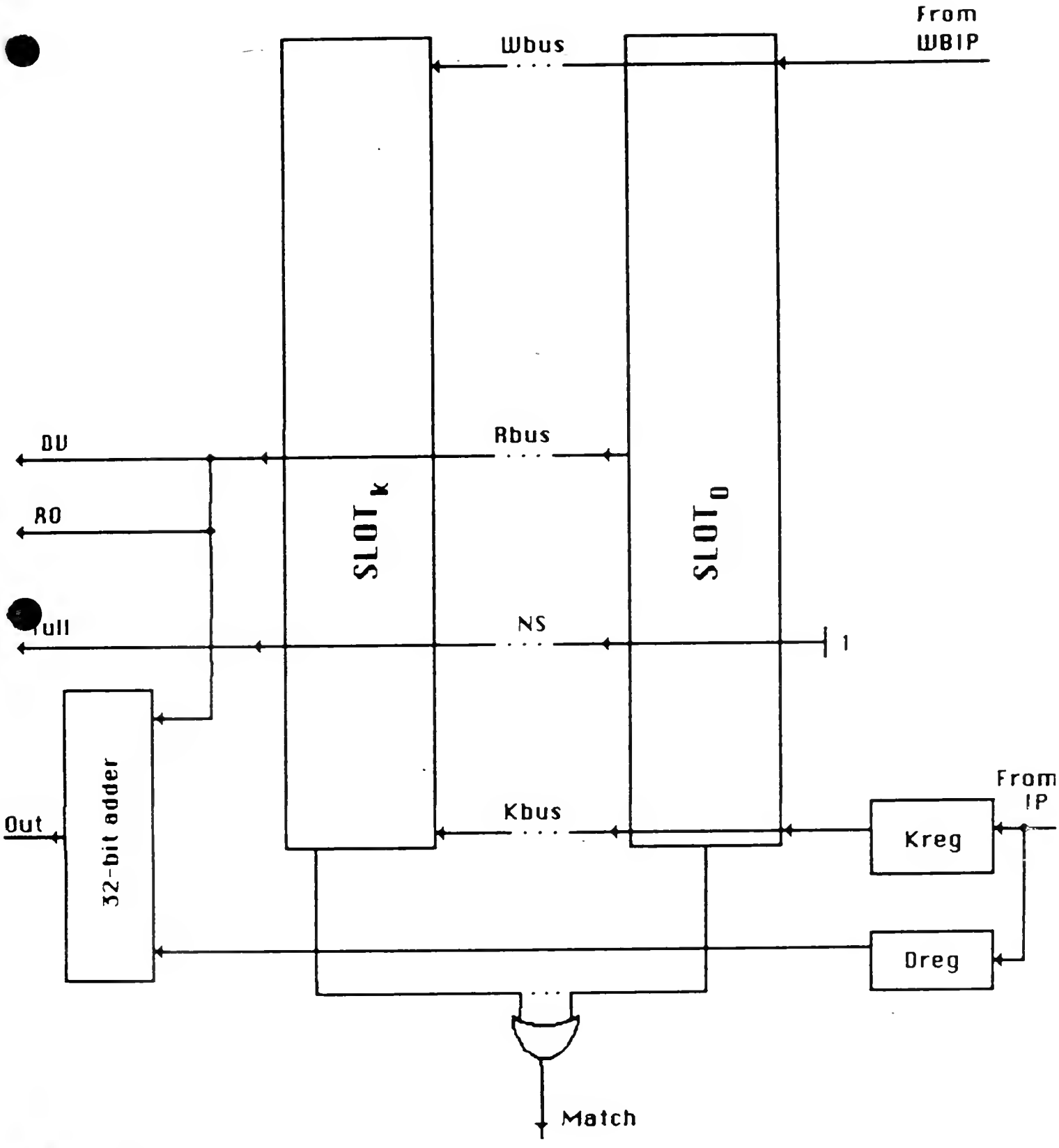

Figure 2 -- Wait Buffer

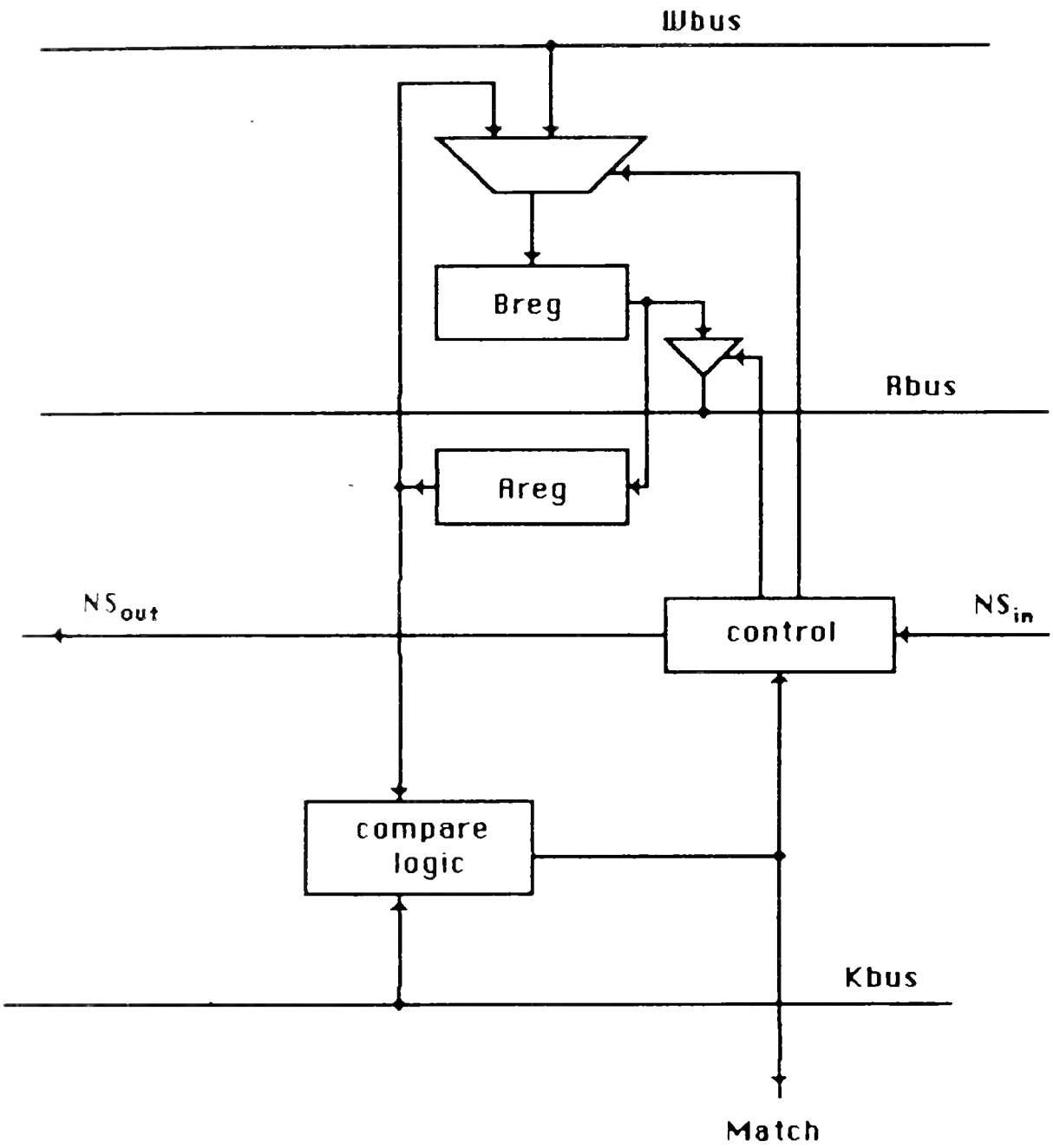

Figure 3 -- Wait Buffer Slot